METHOD OF TRACING DATA TRAFFIC ON A NETWORK

CROSS REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of U.S. Provisional Application No. 60/141,581, filed on June 23, 1999.

TECHNICAL FIELD OF THE INVENTION

This invention relates generally to event tracing and more particularly to a method of tracing data traffic on a network.

BACKGROUND

In this client/server world of computing that we live in today, capacity planning activity is just not limited to the server but to the entire system including the clients, servers and network devices. In the past, network capacity planners have treated the systems without serious concern and system capacity planners have in turn ignored the network devices in their analysis. One major reason for this is the lack of appropriate performance data to tie the two worlds and the fear of overhead of instrumentation.

Accuracy of model input metrics is often cited as the key indicator of the validity of a capacity analysis. In the last couple of years, there have been several papers mentioning the lack of performance data in MICROSOFT WINDOWS NT brand operating system mainly for Capacity Planning. These concerns were addressed by introducing an Event Tracing Facility in the WINDOWS NT 5.0 brand operating system and published the event tracing API. While it addressed system data requirements adequately, data related to network was lacking.

With internet access becoming common place today, there is no drop in the appetite for network bandwidth for web based applications. In fact, high speed networking is a very important focus of WINDOWS NT 5.0 brand operating system which already achieves 1 to 2 Gbps throughput. With network speeds getting this fast, any instrumentation must be highly optimized to take minimal overhead.

Most capacity planning efforts for networks have treated the system as a source generator and focused on frame counts and frame bytes by listening to the wire. Some have employed smart ways of identifying the application responsible for network traffic by scanning the packet headers. These methods are expensive and arbitrary.

SUMMARY OF THE INVENTION

To solve the aforesaid problems, a method of tracing data traffic on a network is provided herein. According to the method, trace instrumentation of the TCP/IP stack provides key information for capacity planners for correctly charging network traffic to the individual services and applications. The TCP/IP stack is instrumented at the transport layer, so that Input/Output Request packets (IRP) representing sends and receives can be detected as they pass through the stack. When such packets are detected an appropriate send or receive event is recorded in a trace log.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 shows the network layers of the WINDOWS NT brand operating system;

Fig. 2 shows the server and workstation services;

Fig. 3 shows the TCP/IP stack;

Fig. 4 is a timeline of a TCP send; and

Fig. 5 is a timeline of a TCP receive.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Event Tracing in WINDOWS NT 5.0 brand operating system

The next version of MICROSOFT WINDOWS NT brand operating system operating system will have a uniform framework for event tracing, specifically for capacity planning. The event tracing mechanism implements a circular buffer pool maintained by the operating system and an API set for Trace Providers, Consumers and Management Control. The trace logger can accept data from kernel mode device drivers and user mode applications.

In addition to providing a facility to log trace data for applications, the WINDOWS NT brand operating system kernel has been instrumented to provide key

capacity planning metrics that were not available through the commonly used performance tool 'Perfmon'. The following system events are instrumented:

1. Process Creation/Deletion event. The ProcessId, parent process Id, Security Id and the Image File name are recorded.

2. Thread Creation/Deletion event. The Thread Id and its process Id are also recorded.

3. Hard Page fault event. The disk signature and the size of the first disk read resulting from the page fault are also recorded.

4. Disk Read/Write event. The disk signature and the size of the operation are recorded.

Multiple logger streams may be active at one time, typically one for the kernel logger and one for each of the trace-enabled applications running on a server. The Consumer APIset makes it easy to process the trace from multiple logger streams in the proper order and returned to the caller one event at a time.

Networking Architecture in WINDOWS NT brand operating system

Networking capabilities are built into WINDOWS NT brand operating system and it is organized as layers as shown in Fig. 1.

WINDOWS NT brand operating system networking components include:

- Transport protocols (DLC 10, NetBEUI 12, NWLink, and TCP/IP 14) define the rules governing communications between two computers.
- Inter-process communication (IPC) components, such as named pipes and mail slots, allow applications to communicate with each other over a network.
- File and Print sharing components allow resources to be made available on a network.

The Multiple uniform naming convention (UNC) Provider (MUP) and Multi-Provider Router (MPR) make it possible to write applications that use a single API to communicate using any network vendor's redirector.

There are two boundary layers in the architecture, namely the Network Driver Interface Specification (NDIS) 16 and Transport Driver Interface (TDI) 18. The NDIS layer provides the interface and a wrapper to the Network Interface Card (NIC) device drivers 20. The TDI boundary layer 18 provides a common interface specification to communicate with various transport drivers. While several protocols are supported in the transport layer, TCP/IP 14 forms the main focal point for all networking activity.

The protocol suite benefits from years of research and is the most favored suite in the Internet. In the WINDOWS NT brand operating system, several services make use of TCP/IP stack, most notably File/Print services and Socket-based applications. Most services require reliable data transmission and use TCP/IP suite for end to end reliable delivery. The File/Print services and Sockets will be explained in more detail in the next section.

File and Print Services

The File and Print services are supported by two services (Redirector and Server) that are layered on top of Transport Driver Interface layer 18 as shown in Figure 2. They provide an encapsulation over the file system and network transparency for applications accessing remote files.

When a process on a WINDOWS NT brand operating system system tries to open a file that resides on a remote computer, the following steps occur:

- The process calls the I/O Manager to request that the file be opened.
- The I/O Manager recognizes that the request is for a file on a remote computer, so it passes it to the redirector file system driver 22 (RDR.Sys).
- The redirector 22 passes the request to lower-level network drivers that transmit it to the remote Server 26 for processing.
- The transport receives a send IO request packet (Irp) for the SMB header and command.
- This send is translated into frames and queued for dispatch on the wire through the Miniport driver.

On the remote WindowsNT server system, when the server service receives a request from a remote computer asking it to read a file that resides on the hard disk, the following steps occur:

- The low-level network drivers receive the request and pass it to the server driver 24 (SRV.SYS).
- The server 26 passes a file read request to the appropriate local file system driver.
- The local file system driver calls lower-level disk device drivers to access the file.
- The data is passed back to the local file system driver.
- The local file system driver passes the data back to the server 26.
- The server 26 passes the data to the lower-level network drivers for transmission back to the client computer 28.

Socket-based Applications

Socket-based applications are supported through the Windows Socket Provider (WinSock). Figure 3 shows the relationship between various modules and TCP/IP.

A Socket based user application that would like to provide a service on port P (pre-advertised) to all clients would open up a socket through WinSock and listen on the socket for connection requests. This would translate as an address object with TCB structures listening on that port with the server's IP address(es) and a wild-card IP address to denote client addresses.

When a client requests a connection, a frame comes in on one of the NICs 28 and 30 (with the client's IP address and connecting port #) and is tied to one of these TCB structures. TCP calls across TDI 18 to indicate to the socket provider, which in turn calls into the User's service application for acceptance. Once accepted, frames are received and sent on the NIC involved in the connection. Though the physical NIC through which the connection data is routed can change over time, the IP addresses and port numbers don't change and lend themselves as connection context for event tracing.

The user application requests a Send to the socket service provider with a pointer to the data. The socket provider, namely AFD 32 will lock the pages in memory and request TCP 14 across TDI 18 to send. TCP 14 cuts the requests to frames and calls the miniport driver 34 corresponding to the NIC through which the data needs to be transmitted. The Miniport driver sends the frames and calls to TCP 14 to complete each frame-send event. The TCP requests are processed asynchronously as a rule, and could happen in the context of the system thread dispatched by the socket provider or a DPC (Deferred Procedure Call) from Ndis.

In the case of receive, the miniport driver 34 services the interrupt and through Ndis 20, queues a DPC to process the frame. This DPC identifies the protocol stack and calls the appropriate Receive Event handler. When this thread executes TCP Receive routine in its context, the data is indicated up or is filled in pre-posted receive buffers from the application.

Event Tracing Sends / Receives

A TCP Send needs to be traced at the end of the send. The end of a send is marked by the processing of an ACK(acknowledgement) from the other connection endpoint corresponding to the last byte of the send. Through TDI, TCP receives an IoRequestPacket (IRP) from AFD with a pointer to a locked user buffer / locked set of pages from a file's cached view. TCP creates a Send-Request structure, which caches this IRP, splits the data into frames, and sends them across. When the last ack (ack acknowledging the last byte sent) arrives, part of receive processing involves queuing the corresponding Send-Request for completion. When the completion queue is processed, the cached IRP is completed to the upper layer.

Between the initiation of the Send and the completion, several copies of the data could get transmitted due to retransmissions, or the send could get cancelled, in which case, a completion doesn't occur. In Fig 4., the timeline of events during a send is explained. Since only the completion is traced, the use of the NIC to send out the said number of bytes is guaranteed. Also, as far as the user application is concerned, the Send through Tdi completed only when the IRP is completed in the socket driver (which may

wake up the blocked thread in the case of synchronous i/o or trigger the appropriate event in the case of async i/o).

Tracing receives is more complex than Sends, in the sense that tracing information needs to be generated at more points than one. In the case of receive, we should not care if the TCP protocol sends out acks or if this is only part of a receive which got cancelled from the other end point. The number of bytes received must be accounted for exactly. We will first take the case of a pre-posted receive and how it is traced for capacity planning purposes.

In the case of pre-posted receives, TCP receives an IRP through TDI to receive a certain size. TCP caches this IRP in a Receive-Request structure. When a chunk of a certain size of data is received (could be less than the requested size, to improve latency) TCP completes the request with the then-available number of bytes and the appropriate buffers. If more needs to be received, the receiver (say application through socket interface) posts more receive-IRPs which are completed as frames are received. In Fig 5., the receive completion and trace timing is explained.

It is possible to receive when no receives are posted. In such cases, the data is indicated to the receiver as soon as the first frame is received. If any more data needs to be received, TCP receives a piggybacked Irp. TCP generates a CP trace in this indicate path to accurately account for the accepted number of bytes.

Data Collection Issues

WINDOWS NT brand operating system uses a packet oriented I/O model for performing I/O operations to disks as well as to network devices. Whenever a user application or service posts a Send/Receive request, an IRP is created and sent to TCP. Typically the IRP is filled with the context of the thread requesting the operation. In the case of sends, it is possible to identify the correct user thread to charge the bandwidth utilization. In the case of receives, a DPC is generated in NDIS, which doesn't run in the context of any user thread. With respect to receives through the indicate-path described above, when the data is accepted without requesting more receives through IRPs, it is not

possible to make a correspondence. In such cases however, the port information that is provided is useful in identifying the service.

What is Instrumented?

1. Send Complete event when a TCP send request is complete (when an ACK for the last byte of the Send Request is received). The source address, destination address, source port number, destination port number, bytes transmitted are also recorded. Events are automatically time-stamped by the trace logger.
2. Receive Indicate event when incoming data is indicated to the upper layers. The source address, destination address, source port and destination port numbers, the size of data received and the process Id of the Process that is being indicated by TCP.
3. Receive Complete event (when a receive-Irp is completed with data). Similar information is collected.

These three trace points cover the majority of the meaningful TCP traffic in the system. It is important to keep in mind that some TCP traffic is not accounted for by this instrumentation. For example, retransmissions from packet loss, receiving IP control msgs (like ICMP etc.).

Instrumentation Overhead

In comparison to other kernel events such as thread create or delete, network events are very high frequency events. As a result, extraordinary care has been taken to minimize the overhead of trace instrumentation. The data being collected is primarily from the Transport Control Block (TCB) structure. The fields in the TCB structure are arranged to make the data relevant to capacity planning in one contiguous block. This allows direct copying from the TCB structure to the Trace Buffers without having to make any intermediary copies. According to measurements obtained during preliminary testing, a network event uses about 128 x86 instructions and logs 24 bytes of data. The actual results may vary, however.

Analysis of sample Traces

Appendix A provides a sample kernel trace fragment from a TCP Send test, translated into readable text format. Each row in the table shows an event instance. Each event instance is described by the fixed header providing the event name, Thread ID that is causing the event, system clock time when the event happened, the kernel and user mode CPU time for the thread. Additional columns in the table show the event specific data associated with each event.

The tests were started after starting up the Trace logger using a command line utility called tracelog.exe. The trace shows the process start and end of the tracelog.exe. Next there is a process start for the TCP send test program (nttcp.exe). Immediately following that, the Tcp Send events can be seen. The source IP address/port and destination IP address/port can also be seen.

The size of transfers is 8K bytes. The thread that's actually performing the send is (thread Id 0, a system thread). However, the process Id that was saved when the connection was created is 1C0, recorded with every send event. This process Id 1C0 corresponds to the nttcp.exe program that initiated the sends. Hence, it can be seen that the network traffic can be charged to processes properly from the traces.

While the TCPSend events triggered on the nttcp connection were in progress, an HTTP request for a webpage happened (shown in italics) and through threadID 39C, this request was handled by the IIS running on port 80 and 3 files were sent out to the HTTP remote browser. The HTTP transaction happened through a keep-alive connection. These events were charged to process 382 (InetInfo.exe).

Possible uses of network traces

Classification by PID:

In summary, since stack event trace generated incorporates PID, it is possible to charge network traffic to a specific process or kernel mode service. Other possible modes of classification, such as per-service, per-NIC, per-remote-request or per-client are indicated in paragraphs below.

Classification of network utilization per Service:

From post-processing the collected trace information, it is possible to classify the bandwidth utilization by application / service. Services and user applications / connections can be characterized using the 4-tuple (SAddr, DAddr, Sport, Dport). The traces collected for the specific port show the utilization for a particular service. From the traces shown, it can be observed that (HTTP) web service active on port 80 can be charged for the receives and sends in italics.

Classification of network utilization per NIC:

Using tools such as IPConfig, it is possible to identify NICs and assigned IP addresses. Parsing the collected trace for a specific IP Source Address gives all the traffic for that particular NIC. It is possible that data rerouting can happen when a transfer is in progress. In such a case, TCP receives an indication and a special stack event trace is generated.

Classification of System resource utilization per remote request / client:

Since Disk I/O events are generated in the context of the process, and remote requests are charged to the same process (through stack event traces), it is possible to identify the running service to charge disk i/o operations to. Based on Destination IP addr, this can be further classified per client of the service.

Conclusion

With the introduction of Event Tracing for WINDOWS NT 5.0 brand operating system, the resource consumption of CPU, Memory (Page faults), Disk I/O and Network and be charged to applications or Services. This will make the task of capacity planning client/server applications running on WINDOWS NT brand operating system servers easier and more accurate.